REMARKS

This paper is being provided in response to the Office Action dated January 2, 2008, for the above-referenced application. In this response, Applicants have amended claims 16 and 37 to clarify that which Applicants consider to be the claimed invention. Applicant respectfully submits that the amendments to the claims are fully supported by the originally filed application.

Applicants thank the examiner for the courtesies extended in connection with the telephonic discussion on December 21, 2007. The amendments herein reflect the points raised in connection with that discussion.

Applicants gratefully acknowledge the indication of allowability of the subject matter of claims 1, 20, 22 and 41.

With respect to the provisional rejections of claims 17, 38, 1, 20, 22, and 41 on the grounds of nonstatutory obviousness-type double patenting over claims 18, 39, 11, and 33 of copending application No. 09/547,550, Applicants respectfully request a response to this issue be deferred until at least one of these applications issues as a U.S. patent, in accordance with M.P.E.P. 804(I)(B)(1).

The rejection of claims 16-17, 19, 37-38, and 40 under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 5,574,898 to Leblang, et al. (hereinafter "Leblang") in view of U.S. Patent No. 5,850,554 to Carver (hereinafter "Carver") is hereby traversed and reconsideration thereof is respectfully requested in view of the amendments to the claims contained herein.

Independent claim 16, as amended herein, recites a computer implemented method for determining a code volatility metric, the method comprising: extracting build information by processing, for at least two builds, one or more software modules produced using a compilation process resulting in the one or more software modules, wherein the build information extracted includes software module information about at least one software module produced as an output of a compilation process for each of the at least two builds; registering the at least two builds by storing the build information corresponding to each of said at least two builds in a database; identifying, by retrieving at least a portion of the build information from the database, a first and a second of the at least two builds; automatically determining software module information about software being tested, where the software module information is gathered from one or more software modules during execution of the software being tested; performing a query of the database to determine code volatility between software modules being tested included in both the first and the second of the at least two builds; and calculating, in response to said query, the code volatility metric using the build information including software module information about the first and the second builds included in the database, the code volatility metric being determined using one or more metrics representing an amount of code change that has occurred between software modules in both the first build and the second build, the one or more metrics including at least one of: a number of functions added, a number of functions removed and a number of functions modified in at least one software module of the second build in comparison to at least one software module of the first build. Claims 17 and 19 depend from independent claim 16.

Independent claim 37, as amended herein, recites a computer readable medium comprising machine executable code stored thereon for determining a code volatility metric, the computer readable medium comprising: machine executable code for extracting build information by processing, for at least two builds, one or more software modules produced using a compilation process resulting in the one or more software modules, wherein the build information extracted includes software module information about at least one software module produced as an output of a compilation process for each of the at least two builds; machine executable code for registering the at least two builds by storing the build information corresponding to each of the at least two builds in a database; machine executable code for identifying, by retrieving at least a portion of the build information from the database, a first and a second of the at least two builds; machine executable code for automatically determining software module information about software being tested, where the software module information is gathered from one or more software modules during execution of the software being tested; machine executable code for performing a query of the database to determine code volatility between software modules being tested included in both the first and the second of the at least two builds; and machine executable code for calculating, in response to said query, said code volatility metric using the build information including software module information about the first and said second builds included in the database, the code volatility metric being determined using one or more metrics representing an amount of code change that has occurred between software modules in both the first build and said second build, the one or more metrics including at least one of: a number of functions added, a number of functions removed and a number of functions modified in at least one software module of the second build in comparison to at least one software module of the first build. Claims 38 and 40 depend directly or indirectly from independent claim 37.

Leblang's Figure 1 includes a version control system and stores of source objects and derived objects. Derived objects are created by running a system build process on a particular version of the source objects (See Figure 1, Col. 5, Lines 36-45). In Figure 7, the version control system is illustrated for use with two developers each using a different view having a set of versions from the versioned object bases (VOBs). (See Figures 2, 7; Col. 6, Lines 8-13; Col. 8, Line 51-Col. 10, Line 9). Source files are the input to the software build process. Equally important are the files that are created by software builds. With the version control system, such files are called derived objects. Each derived object has two main parts, the data itself and an associated configuration record. The configuration record is a "bill of materials" that stores an audit of the build that produced the derived object. This automatically includes a list of the element versions used in the build. It also includes versions of dependencies (such as build tools) that are explicitly declared in the makefile. Derived objects, like elements, can be accessed either with standard UNIX pathnames or with version extended names. (Col. 25, Lines 27-41).

As set forth in the Office Action, Carver discloses reuse of object code based on a persisted version thereof in a database of object modules.

Applicants respectfully submit that neither Leblang, nor Carver, nor any combination thereof show, teach, or suggest a feature of claims 16 and 37, as amended herein, of automatically determining software module information about software being tested, where the software module information is gathered from one or more software modules during execution of the software being tested. Thus, Applicants respectfully submit that Leblang and Carver do not,

alone or in any combination, teach or fairly suggest features as claimed by Applicants. Accordingly, Applicants respectfully request that the rejection be reconsidered and withdrawn.

Based on the above, Applicant respectfully requests that the Examiner reconsider and withdraw all outstanding rejections and objections. Favorable consideration and allowance are earnestly solicited. Should there be any questions after reviewing this paper, the Examiner is invited to contact the undersigned at 508-898-8603.

Respectfully submitted,
MUIRHEAD AND SATURNELLI, LLC

Date: March 28, 2008

Donald W. Muirhead
Reg. No. 33,978

Muirhead and Saturnelli, LLC
200 Friberg Parkway, Suite 1001
Westborough, MA 01581
Tel: (508) 898-8601
Fax: (508) 898-8602

Customer No. 54004